CSCE 421: Spring 2023 Homework 3

Assigned Feb 21, due on Mon, March 2, by 11:59 PM.

Submit your assignments (written+coding) seperately on gradescope. Please name your coding assignment as **'assignment3.py'**. Use the provided python template file, complete ONLY the functions (DO NOT edit function definitions, code outside the function, or use any other libraries).

A Few Notes:

- Coding assignments should be done only in Python.
- Please start early! This includes learning how to use Latex!
- For solution please use the following template `https://www.overleaf.com/latex/templates/neurips-2022/kxymzbjpwsqx`.
- This is an individual assignment. While you are welcome to discuss general concepts together and on the discussion board your solutions must be yours and yours alone.
- **SHOW YOUR WORK.**

Problem 1: Prediction.
Use *Hitters Dataset* provided in *Home work 2*

1. [Code] Fill in the function *read data*, that takes in the filename as string, and returns a pandas dataframe. Hint: you may find read csv function from pandas library useful
2. [Code] Fill in the function *data_preprocess* to complete all preprocessing steps mentioned in *Problem 2* of *Homework 2*. You may use the same code you written in homework 2. *Note: Do not include the Player column.*
3. [Code] Fill in the function *data_split* that given features and labels split the data into a train/test split (in 80% of training and 20% of test ratio). The function returns 4 items in the following order x train, x test, y train, y test.
4. [Code + Written] Fill in the function *train_ridge_regression* to implement a *Ridge Regression Model* for Hitters dataset that returns a dictionary with *lambda_vals* as keys and corresponding *mean accuracy* as value pair. Repeat the training process for *n* times and train model each time for *max_iter* iterations with all *lambda_vals* (Hyper Tuning). Please describe your hyperparameter tuning procedures and optimal *lambda* in *lambda_val* that gives highest accuracy.
5. [Code + Written] Fill in the function *train_lasso* to implement a *Lasso Regularization Logistic Regression* for Hitters dataset that returns a dictionary with *lambda_vals* as keys and corresponding *mean accuracy* as value pair. Repeat the training process for *n* times and train each time for *max_iter* iterations with all *lambda_vals* (Hyper Tuning). Please describe your hyperparameter tuning procedures and optimal *lambda* in *lambda_val* that gives highest accuracy.
6. [Code] Fill in the function *ridge_coefficients* that returns a tuple of trained ridge model with *max_iter* iterations and *alpha* being optimal *lambda_vals* and model coefficients.
7. [Code] Fill in the function *lasso_coefficients* that returns a tuple of trained lasso model with *max_iter* iterations and *alpha* being optimal *lambda_vals* and model coefficients.

8. [Code + Written] Fill in the function *ridge_area_under_curve* that returns area under curve measurements. Plot the ROC curve of the Ridge Model. Include axes labels, legend and title in the Plot. Any of the missing items in plot will result in loss of points.

9. [Code + Written] Fill in the function *lasso_area_under_curve* that returns area under curve measurements. Plot the ROC curve of the Lasso Model. Include axes labels, legend and title in the Plot. Any of the missing items in plot will result in loss of points.

Problem 2: Decision Trees.

For this problem, you'll be coding up regression and classification trees from scratch. Trees are a special class of graphs with only directed edges sans any cycles. They fall under the category of directed acyclic graphs or DAGs. So, trees are DAGs where each child node has only one parent node.

Since trees are easy to design recursively, it is super important that you're familiar with **recursion**. So, it is highly recommended that you brush up on recursion and tree-based search algorithms such as depth-first search (BFS) and breadth-first search (BFS).

- You are **NOT** allowed to use machine learning libraries such as scikit-learn to build regression and classification trees for this assignment.
- You are required to fill out sections of the code marked `"YOUR CODE HERE"`.
- Download the datasets `noisy_sin_subsample_2.csv` here[1] and `new_circle_data.csv` here[2].
- You may add any number of additional supporting functions within functions that you deem necessary.
- Use Node class as a node of the decision trees. DO NOT change the class and function definitions.

Below is a suggested sequence of steps you may want to think along for building regression and classification trees.

1. **Defining a criteria for splitting.**
    1. This criteria assigns a score to a split.
    2. For regression trees, this would be the mean squared error.
    3. For decision trees, this would be the Gini index or entropy.
2. **Create the split.**
    1. Split the dataset by iterating over all the rows and feature columns.
    2. Evaluate all the splits using the splitting criteria.
    3. Choose the best split.
3. **Build the tree.**
    1. Terminal nodes: decide when to stop growing a tree. This would be the maximum allowed depth of the tree or when a leaf is empty or has only 1 element.
    2. Recursive splitting: once a split is created, you can split it further recursively by calling the same splitting function on it.
    3. Building a tree: create a root node and apply recursive splitting on it.
4. **Make predictions with the tree.**
    1. For a given data point, make a prediction using the tree.

(1) Growing a maximum-depth regression tree

The recursive procedure for growing a deep regression tree is illustrated in the figure below. We begin (on the left) by fitting a stump to the original dataset. As we move from left to right the recursion proceeds, with each leaf of the preceding tree split in order to create the next, deeper tree. As can be seen in the

[1]`https://github.com/jermwatt/machine_learning_refined/blob/gh-pages/mlrefined_datasets/nonlinear_superlearn_datasets/noisy_sin_subsample_2.csv`

[2]`https://github.com/jermwatt/machine_learning_refined/blob/gh-pages/mlrefined_datasets/nonlinear_superlearn_datasets/new_circle_data.csv`

rightmost panel, a tree with maximum depth of four is capable of representing the training data perfectly. Fill in the functions marked with YOUR CODE HERE in TreeRegressor class.

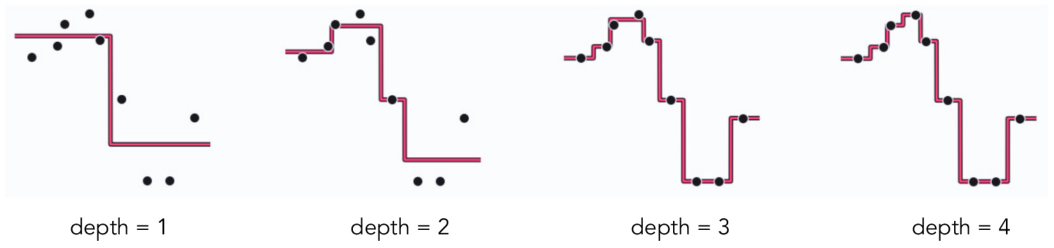

Figure 1: Regressor

(2) Growing a two-class classification tree

The figure above shows the growth of a tree to a maximum depth of seven on a two-class classification dataset. As the tree grows, note how many parts of the input space do not change as leaves on the deeper branches become *pure*. By the time we reach a maximum depth of seven, there is considerable overfitting. Fill in the functions marked with YOUR CODE HERE in TreeClassifier class

Note: function definitions and comments for each function provide a description of the problems the functions are supposed to address.

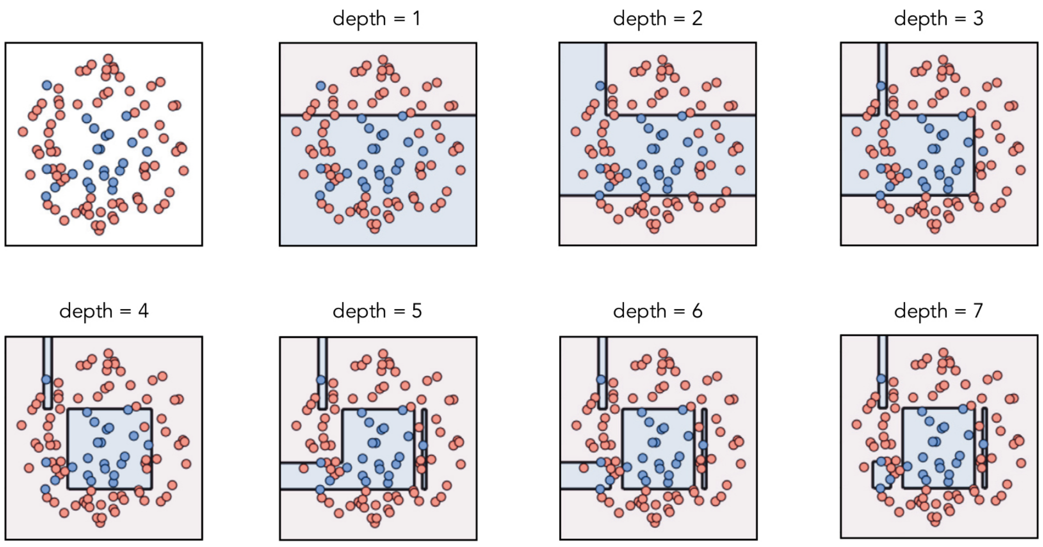

Figure 2: Classifier